In the Claims

1. (Currently amended): A method of ~~instruction execution within~~ operating a microprocessor-based computer system, comprising ~~whereby~~:

~~(a)~~ dividing a sequence of operations from a single execution thread and across multiple executable code regions ~~basic blocks is divided~~ into ~~individual~~ strands;

~~(b)~~ assigning operations ~~instructions~~ from different of said code regions ~~basic blocks are assigned~~ to different of said strands;

~~(c)~~ numbering all of said ~~the~~ strands ~~are numbered~~ at compile time to provide an implicit logical time ordering;

~~(d)~~ explicitly labelling said ~~the~~ operations within each of said ~~individual~~ strands ~~are explicitly labelled~~ with strand numbering and ~~are executed~~ executing them sequentially according to an original sequential order;

~~(e)~~ executing certain operations from ~~different~~ said strands ~~are executed~~ out of order with respect to their said original sequential order;

~~(f)~~ providing each of said strands with ~~has~~ a predication status that determines whether certain operations from ~~the~~ each of said strands should be completed;

~~whereby:~~

composing a plurality of said strands ~~are further composed~~ into an executable code region ~~block~~;

~~means are provided to reset the~~ making a predication status of each ~~individual~~ of said strands resettable at the start of the execution of the executable code region ~~block~~;

giving certain ~~operation~~ of said strands ~~can be given an abort~~ a predication status indicating that certain operations in the executable code region should ~~block could~~ not be completed;

making the predication ~~abort~~ status ~~mechanism may be used~~ usable to support ~~the~~ recovery from data speculative operations between said strands;

repeating execution of the executable code region ~~block should be repeated~~ when a failed data speculation ~~an abort~~ occurs; and

setting the predication status of ~~individual~~ said strands ~~is set~~ upon a repeat execution such that any of said strands that have already been completely executed ~~completed~~ are not re-executed.

2. (Currently amended) The method according to claim 1 whereby operations in one ~~operation~~ of said strands are able to modify the predication status of another of said strands.

Claims 3 to 8 (Cancelled)

9. (Previously presented) The method according to claim 1 whereby said dividing step ~~the subdivision of code into strands~~ is performed from an original sequential stream of instructions.

10. (Cancelled)

11. (Currently amended) The method according to claim 1 whereby each of said ~~operation~~ strands is subdivided into a number of phases according to the type of operations that may be issued and operations that modify processor state that [[is]] are visible outside of the executable ~~block~~ code regions may only be executed in [[the]] a final phase of each of said strands.

12. (Currently amended) The method according to claim 1 whereby said operations from [[the]] said strands may be tagged within their execution word format to indicate ~~the strand~~ to which of said strands they belong.

13. (Currently amended) The method according to claim 12 whereby a tagged strand number is utilized in [[the]] control logic of [[the]] a functional unit to affect [[the]] execution of [[the]] said operations.

14. (Currently amended) The method according to claim 13 whereby attempting execution from a disabled one of said strands ~~substantially~~ disables [[the]] operation of said disabled one of said strands ~~or prevents writeback of results that could affect processor state~~.

15. (Currently amended) The method according to claim 14 whereby the tagging of said operations may be selective and need only ~~necessarily~~ apply to any of said operations that affect processor state that is visible outside of the executable code region ~~block~~.

16. (Currently amended) The method according to claim 14 whereby an execution state of each of said ~~operation~~ strands is distributed to certain functional units by a global bus structure.

17. (Currently amended) The method according to claim 16 whereby the ~~strand~~ execution state of each of said strands is calculated and maintained in a strand control unit.

18. (original) The method according to claim 17 whereby the strand control unit receives requests to modify strand status from one or more functional units.

19. (Currently amended) The method according to claim 1 whereby an abort mechanism is utilized to provide a load speculation mechanism allowing memory loads to be executed earlier than a logically preceding store operation that may access [[the]] a same address.

20. (Currently amended) The method according to claim 19 whereby the load speculation mechanism provides recovery from incorrect speculations by repeat execution of the executable ~~block~~ code region without the requirement for special compensation code.

21. (Previously presented) The method according to claim 19 whereby detection of incorrect load speculations is performed by an explicit functional unit that is used to compare the addresses being used by the load and store operations.

22. (Currently amended) The method according to claim 21 whereby address checks are inserted into the ~~code~~ strands as a result of insertion of ~~such~~ said operations within a graph representation of [[the]] one of said strands built at code generation time.

23. (Currently amended) The method according to claim 1 whereby each of said strands has a committed phase and entry to the committed phase of each strand is performed in the implicit logical time ordering of [[the]] said strands.

24. (Currently amended) The method according to claim 23 whereby an abort of a certain ~~operation~~ of said strands also causes an abort of all of said strands which are logically later.

25. (Currently amended) The method according to claim 23 whereby a branch executed from a particular ~~operation~~ of said strands causes all of said strands which are logically later to be disabled.

26. (Currently amended) The method according to claim 25 whereby branches may be issued out of their original sequential order but are suitably resolved and no actual branch is performed until the end of the executable ~~block~~ code region is reached.

27. (Currently amended) The method according to claim 1 whereby said operations from different of said strands may be interspersed in an execution word for the purposes of improving code density.

28. (Currently amended) The method according to claim 1 whereby an explicit operation may be issued that disables a set of said strands depending on whether they are logically the first being executed.

29. (Currently amended) The method according to claim 10 whereby ~~an operation~~ a strand mechanism is used to convert conditional blocks of code constructed using branches into separate operations strands that do not require a branch.

30. (Currently amended) The method according to claim 1 whereby [[the]] an execution status of said strands upon entry to the executable ~~block~~ code region may be set from a parameter provided by a preceding branch operation.

31. (Currently amended) The method according to claim 30 whereby an entry mechanism may be used to reposition a branch to a logically later of said strands in the ~~block~~ executable code region.

32. (Currently amended) The method according to claim 1 whereby [[the]] scheduling and construction of said strands is influenced by profiling of the executable code region.

33. (Currently amended) The method according to claim 32 whereby a strand ordering of said strands is used to implement static speculations that provides performance benefit whilst seeking to minimize the chances of an incorrect speculation that requires recovery.

34. (Previously presented) A microprocessor for executing instructions using the method of claim 1.

35. (Cancelled)